Empty GIF

The purpose of this module is to provide a directive that serves a *1 x 1* transparent GIF image from the memory. Such files are sometimes used by web designers to tweak the appearance of their website. With this directive, you get an empty GIF straight from the memory instead of reading and processing an actual GIF file from the storage space.

To utilize this feature, simply insert the `empty_gif` directive in the location of your choice:

```
location = /empty.gif {
    empty_gif;
}
```

FLV and MP4

FLV and MP4 are separate modules enabling a simple functionality that becomes useful when serving Flash (FLV) or MP4 video files. It parses a special argument of the request, `start`, which indicates the offset of the section that the client wishes to download or pseudo-stream. The video file must thus be accessed with the following URI: `video.flv?start=XXX`. This parameter is prepared automatically by mainstream video players such as JWPlayer.

This module is not included in the default Nginx build.

To utilize this feature, simply insert the `flv` or `mp4` directive in the location of your choice:

```
location ~* \.flv {
    flv;
}
location ~* \.mp4 {
    mp4;
}
```

Be aware that in case Nginx fails to seek the requested position within the video file, the request will result in a `500 Internal Server Error` HTTP response. JWPlayer sometimes misinterprets this error, and simply displays a *Video not found* error message.

HTTP headers

Two directives are introduced by this module that affect the header of the response sent to the client.

First, `add_header Name value [always]` lets you add a new line in the response headers, respecting the following syntax: `Name: value`. The line is added only for responses with the following codes: `200`, `201`, `204`, `301`, `302`, and `304`. You may insert variables in the `value` argument. If you specify `always` at the end of the directive value, the header will always be added regardless of the response code.

Additionally, the `expires` directive allows you to control the value of the *Expires and Cache-Control HTTP header* sent to the client, affecting the requests of the codes listed previously. It accepts a single value among the following:

- `off`: Does not modify either of the headers
- A time value: The expiration date of the file is set to *the current time* + *the time you specify*. For example, `expires 24h` will return an expiry date set to 24 hours from now
- `epoch`: The expiration date of the file is set to January 1, 1970. The Cache-Control header is set to `no-cache`
- `max`: The expiration date of the file is set to December 31, 2037. The Cache-Control header is set to 10 years

Addition

The Addition module allows you (through simple directives) to add content before or after the body of the HTTP response.

This module is not included in the default Nginx build.

The two main directives are:

```
add_before_body file_uri;
add_after_body file_uri;
```

As stated previously, Nginx triggers a sub-request for fetching the specified URI. Additionally, you can define the type of files to which the content is appended in case your `location` block pattern is not specific enough (default: `text/html`):

```
addition_types mime_type1 [mime_type2…];
addition_types *;
```

Substitution

Along the same lines as that of the preceding module, the Substitution module allows you to search and replace text directly from the response body:

```
sub_filter searched_text replacement_text;
```

This module is not included in the default Nginx build.

Two additional directives provide more flexibility:

- `sub_filter_once` (`on` or `off`, default `on`): Only replaces the text once, and stops after the first occurrence.
- `sub_filter_types` (default `text/html`): Affects the additional MIME types that are eligible for text replacement. The `*` wildcard is allowed.

Gzip filter

This module allows you to compress the response body with the Gzip algorithm before sending it to the client. To enable Gzip compression, use the `gzip` directive (`on` or `off`) at the `http`, `server`, `location`, and even the `if` level (though that is not recommended). The following directives will help you further configure the filter options:

Directive	Description
`gzip_buffers` Context: `http`, `server`, `location`	Defines the number and size of buffers to be used for storing the compressed response. Syntax: `gzip_buffers amount size;` Default: `gzip_buffers 4 4k` (or `8k` depending on the OS).
`gzip_comp_level` Context: `http`, `server`, `location`	Defines the compression level of the algorithm. The specified value ranges from 1 (low compression, faster for the CPU) to 9 (high compression, slower). Syntax: Numeric value. Default: `1`
`gzip_disable` Context: `http`, `server`, `location`	Disables Gzip compression for the requests where the User-Agent HTTP header matches the specified regular expression. Syntax: Regular expression Default: None

Directive	Description
`gzip_http_version` Context: `http`, `server`, `location`	Enables Gzip compression for the specified protocol version. Syntax: `1.0` or `1.1` Default: `1.1`
`gzip_min_length` Context: `http`, `server`, `location`	If the response body length is inferior to the specified value, it is not compressed. Syntax: Numeric value (size) Default: `0`
`gzip_proxied` Context: `http`, `server`, `location`	Enables or disables Gzip compression for the body of responses received from a proxy (see reverse-proxying mechanisms in later chapters). The directive accepts the following parameters; some can be combined: • `off/any`: Disables or enables compression for all requests • `expired`: Enables compression if the *Expires* header prevents caching • `no-cache/no-store/private`: Enables compression if the *Cache-Control* header is set to no-cache, no-store, or private • `no_last_modified`: Enables compression in case the *Last-Modified* header is not set • `no_etag`: Enables compression in case the *ETag* header is not set • `auth`: Enables compression in case an *Authorization* header is set

`gzip_types`
Context: `http`, `server`, `location`
	Enables compression for types other than the default `text/html` MIME type.
	Syntax:
	`gzip_types mime_type1 [mime_type2…];`
	`gzip_types *;`
	Default: `text/html` (cannot be disabled)

`gzip_vary` Context: `http`, `server`, `location`	Adds the *Vary: Accept-Encoding* HTTP header to the response. Syntax: `on` or `off` Default: `off`

Directive	Description
`gzip_window` Context: `http`, `server`, `location`	Sets the size of the window buffer (`windowBits` argument) for Gzipping operations. This directive value is used for calls to functions from the Zlib library. Syntax: Numeric value (size) Default: `MAX_WBITS` constant from the Zlib library
`gzip_hash` Context: `http`, `server`, `location`	Sets the amount of memory that should be allocated for the internal compression state (`memLevel` argument). This directive value is used for calls to functions from the Zlib library. Syntax: Numeric value (size) Default: `MAX_MEM_LEVEL` constant from the Zlib prerequisite library
`postpone_gzipping` Context: `http`, `server`, `location`	Defines a minimum data threshold to be reached before starting the Gzip compression. Syntax: Size (numeric value) Default: `0`
`gzip_no_buffer` Context: `http`, `server`, `location`	By default, Nginx waits until at least one buffer (defined by `gzip_buffers`) is filled with data before sending the response to the client. Enabling this directive disables buffering. Syntax: `on` or `off` Default: `off`

Gzip static

This module adds a simple functionality to the Gzip filter mechanism—when its `gzip_static` directive (`on`, `off`, or `always`) is enabled, Nginx will automatically look for a `.gz` file corresponding to the requested document before serving it. This allows Nginx to send pre-compressed documents instead of compressing documents on the fly at each request. Specifying `always` will force Nginx to serve the gzip version regardless of whether the client accepts gzip encoding.

This module is not included in the default Nginx build.

If a client requests `/documents/page.html`, Nginx checks for the existence of a `/documents/page.html.gz` file. If the `.gz` file is found, it is served to the client. Note that Nginx does not generate `.gz` files itself, even after serving the requested files.

Gunzip filter

With the *Gunzip filter* module, you can decompress a gzip-compressed response sent from the backend in order to serve it *raw* to the client. For example, in cases where the client browser is not able to process the gzipped files (Microsoft Internet Explorer 6), simply insert `gunzip on;` in a location block to employ this module. You can also set the buffer amount and size with `gunzip_buffers amount size;` where `amount` is the amount of buffers to allocate, and `size` is the size of each allocated buffer.

Charset filter

With the *Charset filter* module, you can control the character set of the response body more accurately. Not only are you able to specify the value of the `charset` argument of the Content-Type HTTP header (such as `Content-Type: text/html; charset=utf-8`), but Nginx can also re-encode the data to a specified encoding method automatically.

Directive	Description
`charset` Context: `http`, `server`, `location`, `if`	This directive adds the specified encoding to the Content-Type header of the response. If the specified encoding differs from the `source_charset` one, Nginx re-encodes the document. Syntax: `charset encoding | off;` Default: `off` Example: `charset utf-8;`
`source_charset` Context: `http`, `server`, `location`, `if`	Defines the initial encoding of the response; if the value specified in the `charset` directive differs, Nginx re-encodes the document. Syntax: `source_charset encoding;`
`override_charset` Context: `http`, `server`, `location`, `if`	When Nginx receives a response from the proxy or FastCGI gateway, this directive defines whether or not the character encoding should be checked and potentially overridden. Syntax: `on` or `off` Default: `off`

`charset_types`
Context: `http`, `server`, `location`
	Defines the MIME types that are eligible for re-encoding.
	Syntax:
	`charset_types mime_type1 [mime_type2…];`
	`charset_types * ;`
	Default: `text/html, text/xml, text/plain, text/vnd.wap.wml, application/x-javascript, application/rss+xml`

Directive	Description
`charset_map` Context: `http`	Lets you define character re-encoding tables. Each line of the table contains two hexadecimal codes to be exchanged. You will find re-encoding tables for the `koi8-r` character set in the default Nginx configuration folder (`koi-win` and `koi-utf`). Syntax: `charset_map src_encoding dest_encoding { … }`

Memcached

Memcached is a daemon application that can be connected to via sockets. Its main purpose, as the name suggests, is to provide an efficient distributed key/value memory caching system. The *Nginx Memcached* module provides directives allowing you to configure access to the Memcached daemon.

Directive	Description
`memcached_pass` Context: `location`, `if`	Defines the hostname and port of the Memcached daemon. Syntax: `memcached_pass hostname:port;` Example: `memcached_pass localhost:11211;`
`memcached_bind` Context: `http`, `server`, `location`	Forces Nginx to use the specified local IP address for connecting to the Memcached server. This can come in handy if your server has multiple network cards connected to different networks. Syntax: `memcached_bind IP_address;` Example: `memcached_bind 192.168.1.2;`
`memcached_connect_timeout` Context: `http`, `server`, `location`	Defines the connection timeout in milliseconds (default: 60,000). Example: `memcached_connect_timeout 5000;`
`memcached_send_timeout` Context: `http`, `server`, `location`	Defines the data writing operations timeout in milliseconds (default: 60,000). Example: `memcached_send_timeout 5,000;`
`memcached_read_timeout` Context: `http`, `server`, `location`	Defines the data reading operations timeout in milliseconds (default: 60,000). Example: `memcached_read_timeout 5,000;`
`memcached_buffer_size` Context: `http`, `server`, `location`	Defines the size of the read and write buffer in bytes (default: page size). Example: `memcached_buffer_size 8k;`

Directive	Description
`memcached_next_upstream` Context: `http`, `server`, `location`	When the `memcached_pass` directive is connected to an upstream block (refer to the section on *upstream module*), this directive defines the conditions that should be matched in order to skip to the next upstream server. Syntax: Values selected among `error timeout`, `invalid_response`, `not_found`, or `off` Default: `error timeout` Example: `memcached_next_upstream off;`
`memcached_gzip_flag` Context: `http`, `server`, `location`	Checks for the presence of the specified flag in the memcached server response. If the flag is present, Nginx sets the `Content-encoding` header to `gzip` to indicate that it will be serving gzipped content. Syntax: numeric flag Default: (none) Example: `memcached_gzip_flag 1;`

Additionally, you will need to define the `$memcached_key` variable, which defines the key of the element that you are placing or fetching from the cache. You may, for instance, use `set $memcached_key $uri` or `set $memcached_key $uri?$args`.

Note that the Nginx Memcached module is only able to retrieve data from the cache; it does not store the results of requests. Storing data in the cache should be done by a server-side script. You just need to make sure to employ the same key-naming scheme in both your server-side scripts and the Nginx configuration. As an example, we could decide to use `memcached` to retrieve data from the cache before passing the request to a proxy if the requested URI is not found (see *Chapter 7, From Apache to Nginx*, for more details about the Proxy module):

```
server {
    server_name example.com;
    […]
    location / {
        set $memcached_key $uri;
        memcached_pass 127.0.0.1:11211;
        error_page 404 @notcached;
    }
    location @notcached {
        internal;
```

```
            # if the file is not found, forward request to proxy
            proxy_pass 127.0.0.1:8080;
        }
    }
```

Image filter

This module provides image processing functionalities through the *GD Graphics Library* (also known as *gdlib*).

This module is not included in the default Nginx build.

Make sure to employ the following directives on a `location` block that filters image files only, such as `location ~* \.(png|jpg|gif)$ { … }`.

Directive	Description
`image_filter` Context: `location`	Lets you apply a transformation on the image before sending it to the client. There are five options available: • `test`: Makes sure that the requested document is an image file, returns a `415 Unsupported media type` HTTP error if the test fails. • `size`: Composes a simple JSON response indicating information about the image such as the size and type (for example, `{ "img": { "width":50, "height":50, "type":"png"} }`). If the file is invalid, a simple `{}` is returned. • `resize width height`: Resizes the image to the specified dimensions. • `crop width height`: Selects a portion of the image of the specified dimensions. • `rotate 90 | 180 | 270`: Rotates the image by the specified angle (in degrees). Example: `image_filter resize 200 100;`
`image_filter_buffer` Context: `http`, `server`, `location`	Defines the maximum file size for the images to be processed. Default: `image_filter_buffer 1m;`

Directive	Description
`image_filter_jpeg_quality` Context: `http`, `server`, `location`	Defines the quality of the output JPEG images. Default: `image_filter_jpeg_quality 75;`
`image_filter_transparency` Context: `http`, `server`, `location`	By default, PNG and GIF images keep their existing transparency during the operations that you perform by using the Image Filter module. If you set this directive to `off`, all existing transparency will be lost, but the image quality will be improved. Syntax: `on` or `off` Default: `on`
`image_filter_sharpen` Context: `http`, `server`, `location`	Sharpens the image by the specified percentage (value may exceed 100). Syntax: Numeric value Default: `0`
`image_filter_interlace` Context: `http`, `server`, `location`	Enables interlacing of the output image. If the output image is a JPG file, the image is generated in the *progressive JPEG* format. Syntax: `on` or `off` Default: `off`

Please note that when it comes to JPG images, Nginx automatically strips off the metadata (such as EXIF) if it occupies more than five percent of the total space of the file.

XSLT

The Nginx XSLT module allows you to apply an XSLT transform on an XML file or response received from a backend server (proxy, FastCGI, and so on) before serving the client.

This module is not included in the default Nginx build

Directive	Description
`xml_entities` Context: `http`, `server`, `location`	Specifies the DTD file containing symbolic element definitions. Syntax: File path Example: `xml_entities xml/entities.dtd;`

Directive	Description
`xslt_stylesheet` Context: `location`	Specifies the XSLT template file path with its parameters. Variables may be inserted in the parameters. Syntax: `xslt_stylesheet template [param1] [param2...];` Example: `xslt_stylesheet xml/sch.xslt param=value;`

`xslt_types`
Context: `http`, `server`, `location`
	Defines the additional MIME types, other than `text/xml`, to which the transforms may apply.
	Syntax: MIME type
	Example:
	`xslt_types text/xml text/plain;`
	`xslt_types *;`

`xslt_paramxslt_string_param` Context: `http`, `server`, `location`	Both the directives allow defining parameters for XSLT stylesheets. The difference lies in the way the specified value is interpreted: the XPath expressions in the value are processed using `xslt_param`, while `xslt_string_param` is used for plain character strings. Syntax: `xslt_param key value;`

About your visitors

The following set of modules provides extra functionality that helps you find out more information about the visitors by parsing client request headers for browser name and version, assigning an identifier to requests presenting similarities, and so on.

Browser

The Browser module parses the User-Agent HTTP header of the client request in order to establish values for the variables that can be employed later in the configuration. The three variables produced are:

- `$modern_browser`: If the client browser is identified as being a modern web browser, the variable takes the value defined by the `modern_browser_value` directive.
- `$ancient_browser`: If the client browser is identified as being an old web browser, the variable takes the value defined by `ancient_browser_value`.
- `$msie`: This variable is set to `1` if the client is using a Microsoft IE browser.